CMPS 312 Project Phase 2 – Data Management using Firestore, Firebase Storage and local SQLite Database (10% of the course grade)

QAR QAR QAR QAR QAR QAR

The project phase 2 submission is due by 12noon Sunday 28th November 2021. Demos will be organized on the same day.

1. Deliverables

You are required to complete the implementation delivery of YalaPay for managing the payments of invoices. Its purpose is to streamline the financial transactions between a company and its customers by speeding the collection of payments from customers. YalaPay also manages the cashing cheques and provides valuable financial reports to track pending payments.

In this phase, you will extend phase 1 app solution. You are required to manage the app data using Firestore, Firebase Storage and local SQLite Database. Your project phase 2 deliverables include:

Part 1 - Firestore Database Design and Implementation

1. Design and implement Firestore database to manage YalaPay app data.
2. Implement the repository methods to read/write entities using Firbase Firestore as the data source. All **data filtering should be done on the server** using Firestore queries and only the required data should be retrieved.
   Also, it is important to ensure that the list of displayed Invoices, Payments and Cheque Deposits are auto-refreshed whenever the data is updated in the online Firestore.

   While using the app the data related to Invoices, Payments and Cheque Deposits must be read and written to FireStore (no need to store such data in the local SQLite database).

   All other data such as Customers, Return Reasons and other reference data should read/written from a local YalaPay SQLite database to lower the cost of using FireStore. These data should be also stored in FireStore and synced upon request to SQLLite database (see Part 2 for further details).
3. When adding/updating a Cheque payment to Firestore you need to upload the associated cheque image to Firebase Storage under cheques subfolder. The cheuque image name should be ChequeNo (e.g., *2468.jpg*). Also, when a cheque payment is delete then delete its associated cheque image from Firebase Storage.
4. Provide ***Initialize Firebase Database*** menu option in the Navigation Drawer. When clicked, if Firestore database is empty, initialize Firestore database with data from json files such as *customers.json*, *return-reasons.json*, *invoices.json*, *payments.json*, *cheque-deposits.json*, *etc*.
5. Document your database design in a schema diagram.

Important notes: When adding all entities (e.g., Customer, Invoice, Payment, Cheque Deposit, Customer) to Firestore you should let Firestore auto-set the **id**.

Part 2 - Sync Reference Data from Firestore to a local YalaPay SQLite database

To lower the cost of using FireStore, the app **reference data** (i.e., data *other than Invoices, Payments and Cheque Deposits)* such as Customers, Return Reasons and other reference data should read and written to a local YalaPay SQLite database.

1. Implement the needed entity annotations to manage the app reference data in a local SQLite database using Room (no need to store data related to *Invoices, Payments and Cheque Deposits* in the local YalaPay SQLite database).
2. Implement the Data Access Objects (DAO) and repositories to read/write from SQLite using the Room library.
3. When the app starts if the local SQLite database is empty then the app should first populate the local database using the data from FireStore.
4. The app should provide a menu item in the Navigation Drawer to ***Sync Data from FireStore*** to the local SQLite database upon request. A version number could be used to manage the synchronization. Only entries from FireStore with a higher version number should be synced with corresponding entries in the local database (e.g., if a Customer 123 from FireStore has version 2 while the same Customer 123 has version 1 in the local database then the customer details from FireStore should be used to update the customer details in the local database. If the versions are the same then no update should take place). Also, new data entries (having version set to 0) and not present in the local datastore should be inserted.
5. When a Customer is created using the App, it should be written to Firestore first then it should be added to the local SQLite database.
6. Document your SQLite database design in a schema diagram.

Part 3 – Signup and Signin using Firebase Authentication

1. To use the app the user must authenticate first. If the user was not authenticated before the app should display the login screen when the app starts.
2. Implement signin using Firebase Authentication.
3. Implement sign-up, the app should provide a signup screen to the enter the user details. Upon submission, a user account should be created on Firebase Authentication (using the user email and password) and the user details should be stored in a **users** collection in Firestore. The user details should include: email (should be used as the document id), firstName, lastName, phone, address (buildingNo, street and city), photoUri, and role. The user role could be either Employee or Manager. They both have the same access except the repots are only accessible by the Manager.
4. When creating or updating a user, the app should allow the user to select an existing user profile photo or take a photo using the phone camera. Then it should upload the selected photo to Firebase Storage. The photo name could be set to *UUID.randomUUID().toString()*. The Uri of the uploaded profile photo should be recorded in the user document on FireStore.
5. The app should provide a **Signout** menu item in the Navigation Drawer to allow the user the signout and navigate to the Login screen.

Part 4 - Design and Testing Documentation

1. Document 4 **technical** lessons learned by comparing your submitted project phase 1 with the model solution provided. You need to provide detailed reflections about the new concepts and technical lessons learnt.
2. Document the MVVM architecture diagram for your overall solution design.
3. Firestore database schema diagram and SQLite database schema diagram.
4. Write a testing document including screenshots of conducted tests illustrating a working implementation.
5. Every team member should submit a description of their project contribution. Every team member should participate in the solution demo and answer questions during the demo.

Important notes:

- Continue posting your questions to https://github.com/cmps312f21/cmps312-content/issues

- Do not forget to submit your design and testing document (in Word format) and fill the *Functionality* column of the grading sheet provided in the phase 2 Word template.

- Push your implementation and documentation to your group GitHub repository as you make progress.

- You need to test as you go!
- Seek further clarification about the requirements/deliverables during office hours or by posting questions online.

2. Grading rubric

Criteria	%	Functionality*	Quality of the implementation
1) Cloud Firestore Database Design and Implementation Repositories to interact with Firestore	50		
2) Sync Reference Data from Firestore to a local YalaPay SQLite database	30		
3) Signup and Signin using Firebase Authentication	10		
4) Design and Testing Documentation *** Design documentation:** - 4 key lessons learned from Phase 1. - MVVM architecture diagram. - Firestore database schema diagram and SQLite database schema diagram. *** Testing documentation:** with evidence of working implementation using snapshots illustrating the results of your solution testing (you must use the provided template).	10		
6) **Discussion of the project contribution** of each team member [-10pts if not done]			
Total	100		
Copying and/or plagiarism or not being able to explain or answer questions about the implementation	-100		

* **Possible grading for functionality** - ***Working*** (get 70% of the assigned grade), ***Not working*** (lose 40% of the assigned grade and ***Not done*** (get 0). The remaining grade is assigned to the quality of the implementation.

In case your implementation is not working then 40% of the grade will be lost and the remaining 60% will be determined based on the code quality and how close your solution to the working implementation.

Solution quality also includes meaningful naming of identifiers (according to Android naming conventions), no redundant code, simple and efficient design, clean implementation without unnecessary files/code, use of comments where necessary, proper code formatting and indentation.

Marks will be reduced for code duplication, poor/inefficient coding practices, poor naming of identifiers, unclean/untidy submission, and unnecessary complex/poor user interface design.